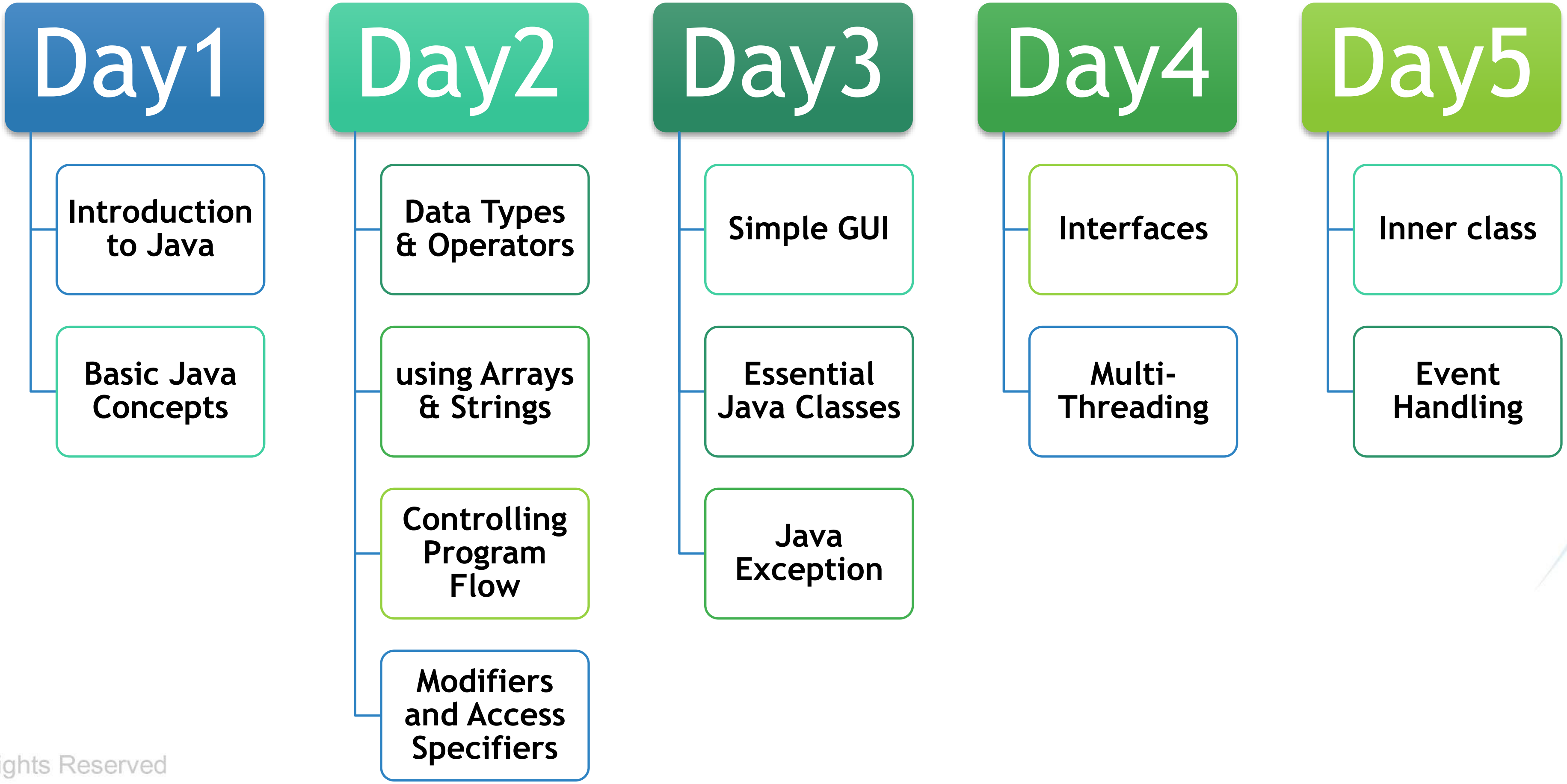
Course Outline

☑ DONE



Day1
- Introduction to Java
- Basic Java Concepts

Day2
- Data Types & Operators
- using Arrays & Strings
- Controlling Program Flow
- Modifiers and Access Specifiers

Day3
- Simple GUI
- Essential Java Classes
- Java Exception

Day4
- Interfaces
- Multi-Threading

Day5
- Inner class
- Event Handling

ITI - Jets © 2018 All Rights Reserved

Java Programming

Data Types & Operators

Java™ Education and Technology Services

Presented By
Dr. Eman Hesham, DBA.

Invest In Yourself ,
Develop Your Career

ITI - Jets © 2018 All Rights Reserved

Agenda

I. Data Types & Operators

1. Data Types
2. Wrapper Classes
3. Reference Data types
4. Operators

Agenda

I. Data Types & Operators

1. Data Types
2. Wrapper Classes
3. Reference Data types
4. Operators

Identifiers

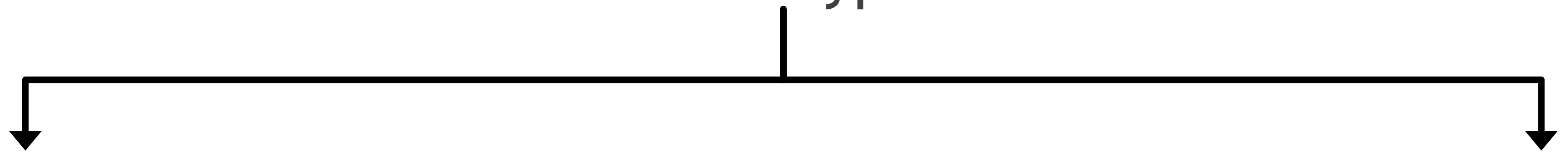
- An identifier is the name given to a feature (variable, method, or class).

- An identifier can begin with either:
  - a letter,
  - $, or
  - underscore.

- Subsequent characters may be:
  - a letter,
  - $,
  - underscore, or
  - digits.

Data types

- Data types can be classified into two types:



Primitive

Boolean	boolean	1 bit	(true/false)
Integer	byte	1 B	$(-2^{7} \rightarrow 2^{7}-1)$ $(-128 \rightarrow +127)$
	short	2 B	$(-2^{15} \rightarrow 2^{15}-1)$ (-32,768 to +32,767)
	int	4 B	$(-2^{31} \rightarrow 2^{31}-1)$
	long	8 B	$(-2^{63} \rightarrow 2^{63}-1)$
Floating Point	float	4 B	Standard: IEEE 754 Specification
	double	8 B	Standard: IEEE 754 Specification
Character	char	2 B	unsigned Unicode chars $(0 \rightarrow 2^{16}-1)$

Reference

- Arrays
- Classes
- Interfaces

ITI - Jets © 2018 All Rights Reserved

Literals

- A literal is any value that can be assigned to a primitive data type or String.

boolean	true false	
char	‘a’ …. ’z’ ‘A’ …. ‘Z’	
	‘\u0000’ …. ‘\uFFFF’	
	‘\n’ ‘\r’ ‘\t’	
Integral data type	15	Decimal (int)
	15L	Decimal (long)
	017	Octal
	0XF	Hexadecimal
Floating point data type	73.8	double
	73.8F	float
	5.4 E-70	$5.4 * 10^{-70}$
	5.4 e+70	$5.4 * 10^{70}$

ITI - Jets © 2018 All Rights Reserved

Agenda

I. Data Types & Operators
  1. Data Types
  2. Wrapper Classes
  3. Reference Data types
  4. Operators

Wrapper Classes

- Each primitive data type has a corresponding wrapper class.

boolean	→	Boolean
byte	→	Byte
char	→	Character
short	→	Short
int	→	Integer
long	→	Long
float	→	Float
double	→	Double

ITI - Jets © 2018 All Rights Reserved

Wrapper Classes

- There are three reasons that you might use a wrapper class rather than a primitive:

1. As an argument of a method that expects an object.
2. To use constants defined by the class,
   - such as MIN_VALUE and MAX_VALUE, that provide the upper and lower bounds of the data type.
3. To use class methods for
   - converting values to and from other primitive types,
   - converting to and from strings,
   - converting between number systems (decimal, octal, hexadecimal, binary).

Wrapper Classes

- They have useful methods that perform some general operation, for example:

`primitive xxxValue()`	→	convert wrapper object to primitive
`primitive parseXXX(String)`	→	convert String to primitive
`Wrapper valueOf(String)`	→	convert String to Wrapper

```
Integer i2 = new Integer(42);
byte b = i2.byteValue();
double d = i2.doubleValue();
```

```
String s3 = Integer.toHexString(254);
 System.out.println("254 is " + s3);
```

Wrapper Classes

- They have special static representations, for example:

POSITIVE_INFINITY		In class Float & Double
NEGATIVE_INFINITY		
NaN	Not a Number	

Agenda

I. Data Types & Operators

1. Data Types
2. Wrapper Classes
3. **Reference Data types**
4. Operators

Reference Data types: Classes

- General syntax for creating an object:

```
MyClass myRef;        // just a reference
myRef = new MyClass(); // construct a new object
```

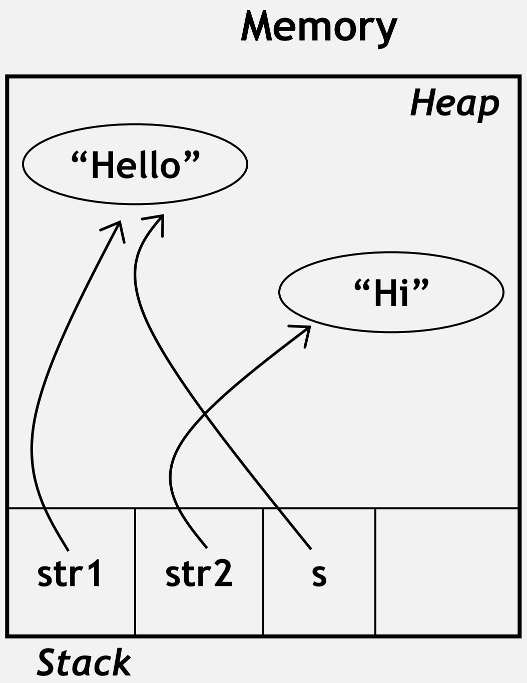
- Or on one line:

```
MyClass myRef = new MyClass();
```

- An object is garbage collected when there is no reference pointing to it.

Reference Data types: Classes

```
String str1;                // just a null reference
str1 = new String(“Hello”);  // object construction

String str2 = new String(“Hi”);

String s = str1;     //two references to the same object

str1 = null;

s = null;    // The object containing “Hello” will
             // now be eligible for garbage collection.

str1.anyMethod();   // ILLEGAL!
             //Throws NullPointerException
```



ITI - Jets © 2018 All Rights Reserved

Agenda

I. Data Types & Operators

1. Data Types
2. Wrapper Classes
3. Reference Data types
4. Operators

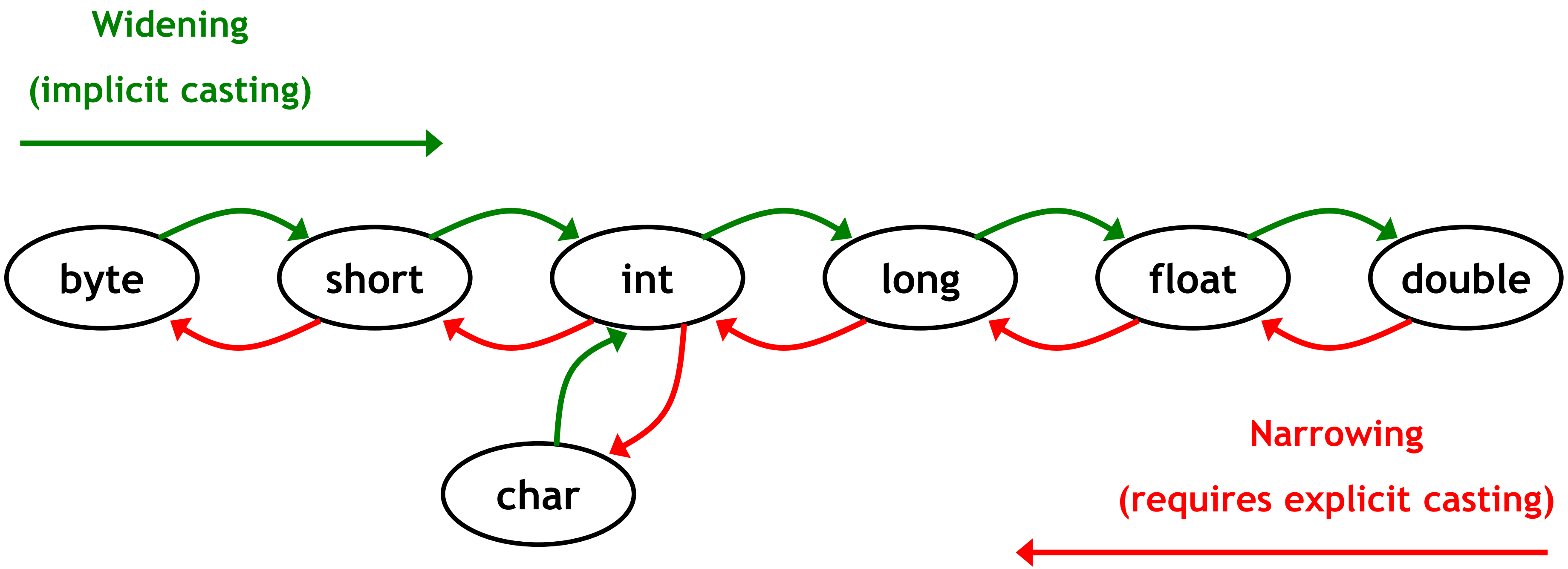
Operators

- Operators are classified into the following categories:
  - Unary Operators.
  - Arithmetic Operators.
  - Assignment Operators.
  - Relational Operators.
  - Shift Operators.
  - Bitwise and Logical Operators.
  - Short Circuit Operators.
  - Ternary Operator.

Operators

- Unary Operators:

+	-	++	--	!	~	( )
positive	negative	incremen t	decrement	boolean complement	bitwise inversion	casting



ITI - Jets © 2018 All Rights Reserved

Operators

- Arithmetic Operators:

+	-	*	/	%
add	subtract	multiply	division	modulo

- Assignment Operators:

=	+=	-=	*=	/=	%=	&=	|=	^=

- Relational Operators:

<	<=	>	>=	==	!=	Instanceof

Operations must be performed on homogeneous data types

ITI - Jets © 2018 All Rights Reserved

Operators

- Bitwise and Logical Operators:

&	|	^
AND	OR	XOR

- Short Circuit Operators:

&&	||
(condition1 AND condition2)	(condition1 OR condition2)

Operators

- Ternary Operator:

```
condition ?true statement:false statement
```

```
int y = 15;                              If(y<z)
int z = 12;                  →                x=10;
int x = y<z? 10 : 11;                    else
                                              x=11;
```

Java Programming
Using Arrays & Strings

Java™ Education
and Technology Services

Invest In Yourself ,
Develop Your Career

ITI - Jets © 2018 All Rights Reserved

What is Array?

- An Array is a collection of variables of the same data type.
- Each element can hold a single item.
- Items can be primitives or object references.
- The length of the array is determined when it is created.

What is Array?

- Java Arrays are homogeneous.
- You can create:
  - An array of primitives,
  - An array of object references, or
  - An array of arrays.
- If you create an array of object references, then you can store subtypes of the declared type.

Declaring an Array

- General syntax for creating an array:

```
Datatype[]  arrayIdentifier;                // Declaration
arrayIdentifier = new Datatype [size];  // Construction
```

- Or on one line, hard coded values:

```
Datatype[] arrayIdentifier = { val1, val2, val3, val4 };
```

- To determine the size (number of elements) of an array at runtime, use:

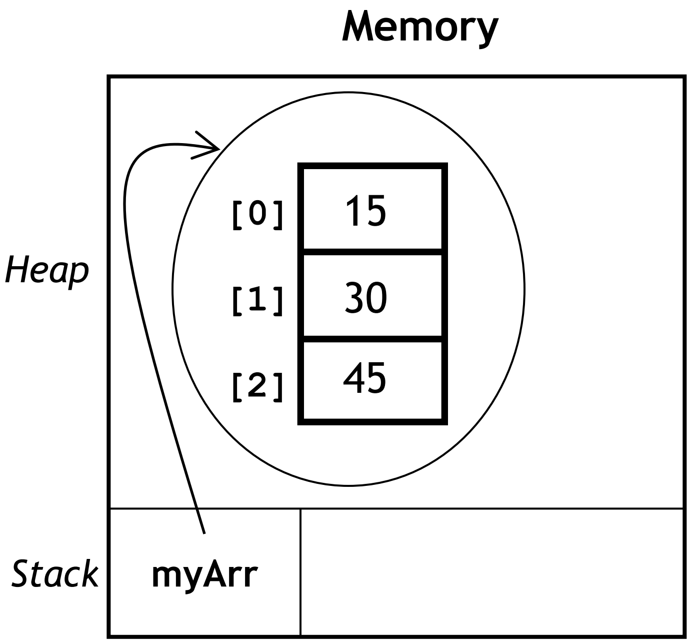
```
        arrayIdentifier.length
```

ITI - Jets © 2018 All Rights Reserved

Declaring an Array

- **Example1:** Array of Primitives:

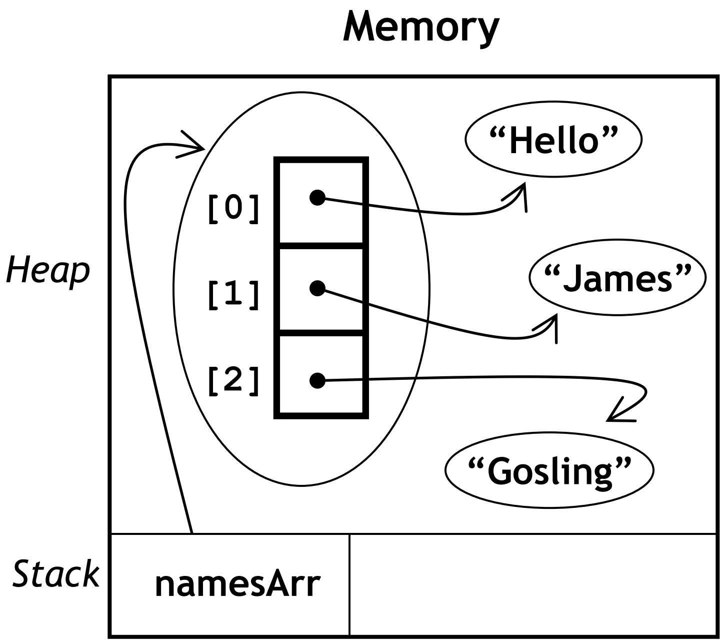
```
int[] myArr;

myArr = new int[3];

myArr[0] = 15 ;
myArr[1] = 30 ;
myArr[2] = 45 ;

System.out.println(myArr[2]);


myArr[3] = … ; // ILLEGAL!
    //Throws ArrayIndexOutOfBoundsException
```



Declaring an Array

- **Example2:** Array of Object References:

```
String[] namesArr;

namesArr = new String[3];

namesArr[0].anyMethod()  // ILLEGAL!
                         //Throws NullPointerException

namesArr[0] = new String(“Hello”);
namesArr[1] = new String(“James”);
namesArr[2] = new String(“Gosling”);

System.out.println(namesArr[1]);
```



ITI - Jets © 2018 All Rights Reserved

String Operations

- Although String is a reference data type (class),
  - it may figuratively be considered as the 9th data type
    because of its special syntax and operations.
  - Creating String Object:

```
String myStr1 = new String(“Welcome”);
String sp1 = “Welcome”;
String sp2 = “ to Java”;
```

  - Testing for String equality:

```
if(myStr1.equals(sp1))

if(myStr1.equalsIgnoreCase(sp1))

if(myStr1 == sp1)
 // Shallow Comparison (just compares the references)
```

ITI - Jets © 2018 All Righ

Strings Operations

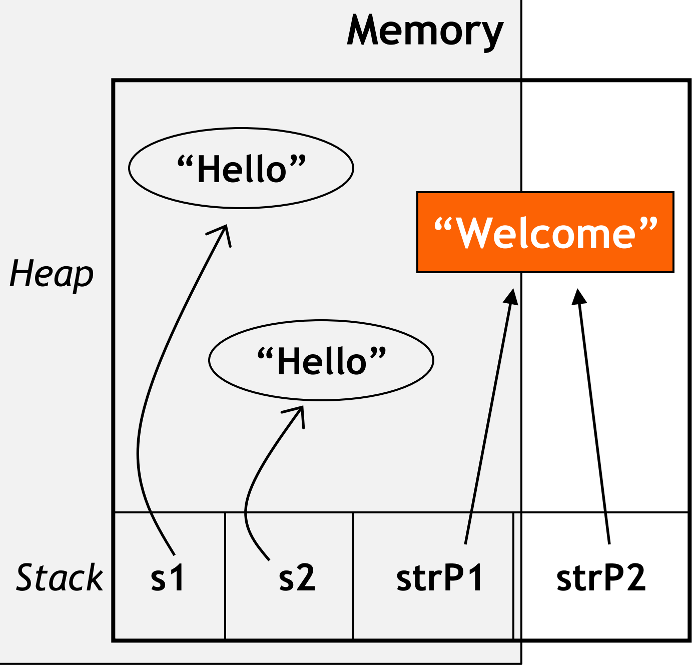
- The ‘+’ and ‘+=‘ operators were overloaded for class String to be used in concatenation.

```
String str = myStr1 + sp2;           // “Welcome to Java”
str += “ Programming”;               // “Welcome to Java Programming”
str = str.concat(“ Language”);  // “Welcome to Java Programming Language”
```

- Objects of class String are immutable
  - you can’t modify the contents of a String object after construction.
- Concatenation Operations always return a new String object that holds the result of the concatenation. The original objects remain unchanged.

String Pool

```
String s1 = new String(“Hello”);
String s2 = new String(“Hello”);


String strP1 = “Welcome” ;
String strP2 = “Welcome” ;
```



- String objects that are created without using the “new” keyword are said to belong to the “String Pool”.

String Pool

- String objects in the pool have a special behavior:
  - If we attempt to create a fresh String object with exactly the same characters as an object that already exists in the pool (case sensitive), then no new object will be created.
  - Instead, the newly declared reference will point to the existing object in the pool.
- Such behavior results in a better performance and saves some heap memory.
- Remember: objects of class String are immutable.

Java Programming

Controlling Program Flow

Java™ Education
and Technology Services

Invest In Yourself ,
Develop Your Career

ITI - Jets © 2018 All Rights Reserved

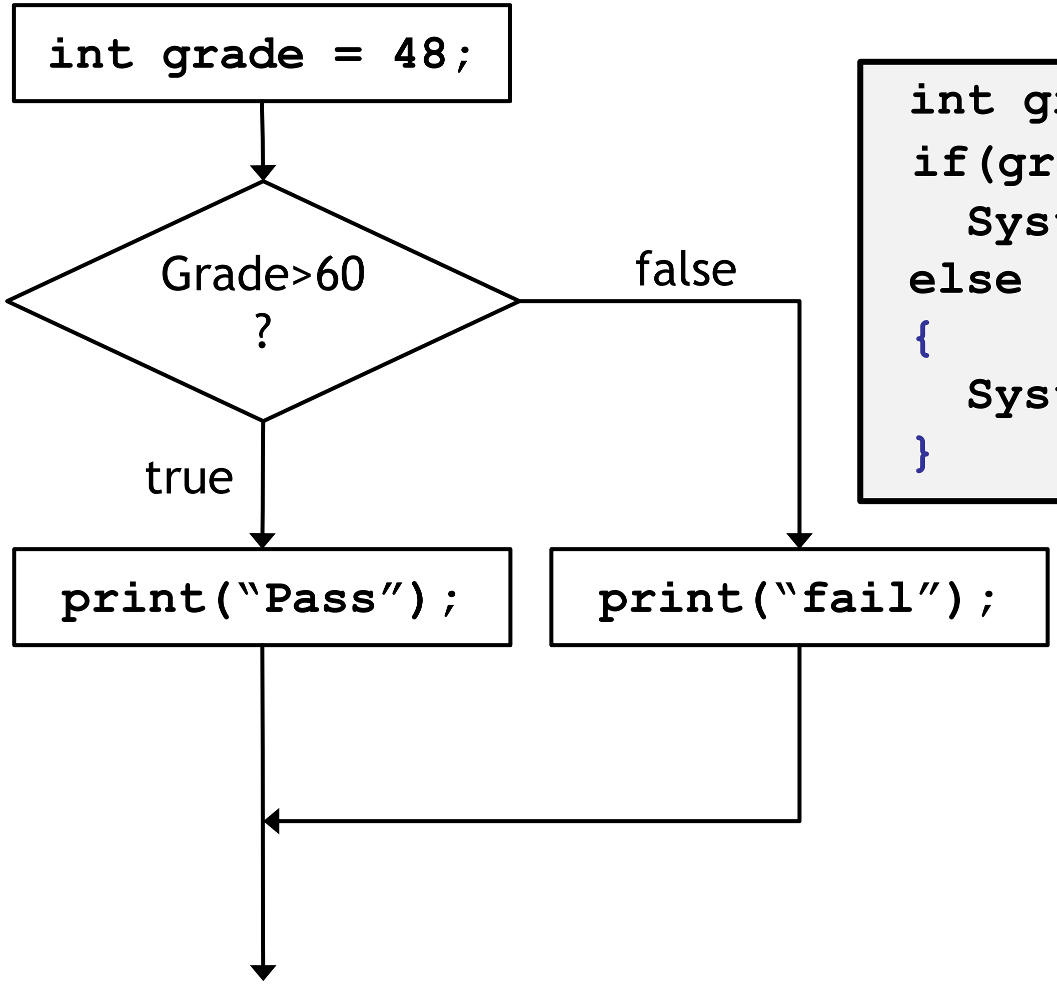
Flow Control: Branching - if, else

- The if and else blocks are used for binary branching.
- **Syntax:**

```
if(boolean_expr)
{
        …
        …    //true statements
        …
}
[else]
{
        …
        …    //false statements
        …
}
```

ITI - Jets © 2018 All Rights Reserved

if, else Example

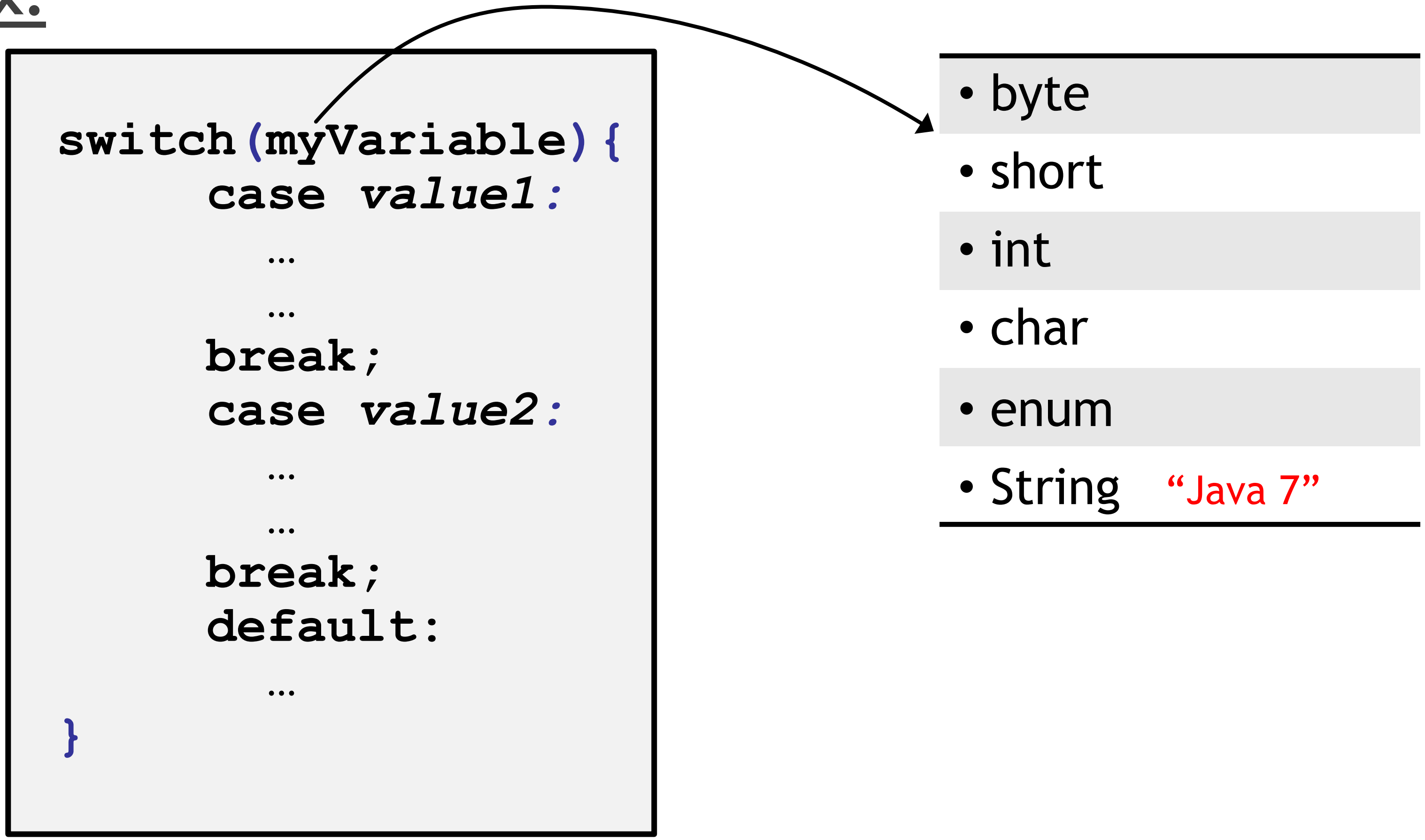


```
int grade = 48;
if(grade > 60)
  System.out.println(“Pass”);
else
{
  System.out.println(“Fail”);
}
```

ITI - Jets © 2018 All Rights Reserved

Flow Control: Branching - switch

- The switch block is used for multiple branching.
- **Syntax:**



```
switch(myVariable){
     case value1:
       …
       …
     break;
     case value2:
       …
       …
     break;
     default:
       …
}
```

- byte
- short
- int
- char
- enum
- String "Java 7"

ITI - Jets © 2018 All Rights Reserved

Flow Control: Branching – switch (EX.)

```
public class StringSwitchDemo {
 public int getMonthNumber(String month) {
  int monthNumber = 0;
  switch (month.toLowerCase()) {
   case "january":
           monthNumber = 1;
           break;
   case "february":
           monthNumber = 2;
           break;
   .......
   default:
           monthNumber = 0;
           break;
  } return monthNumber;
 }}
```

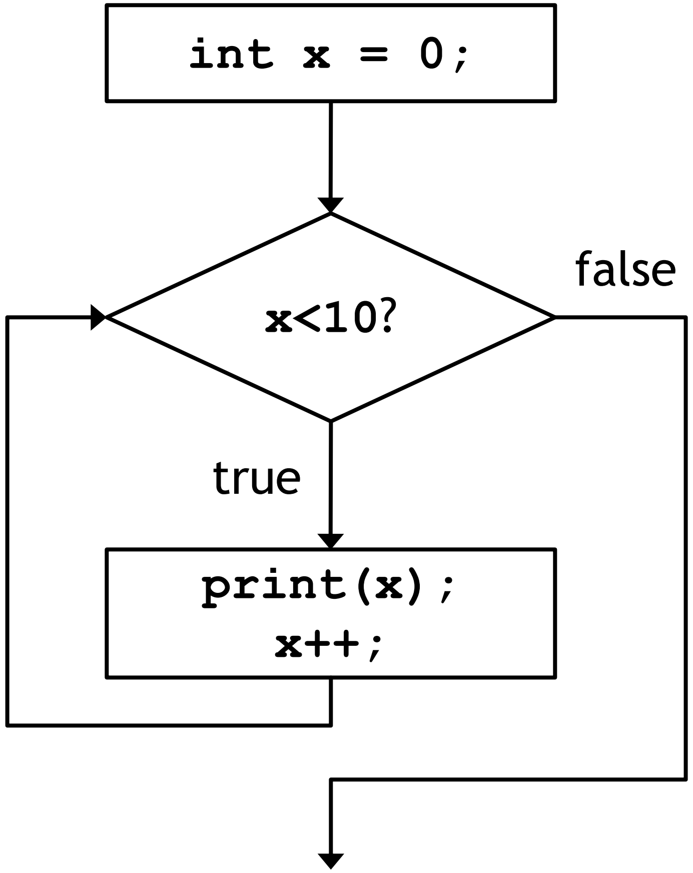
Flow Control: Iteration – while loop

- The while loop is used when the termination condition occurs unexpectedly and is checked at the beginning.
- **Syntax**:

```
while (boolean_condition)
{
        …
        …
        …
}
```

ITI - Jets © 2018 All Rights Reserved

while loop Example



```
int x = 0;
while (x<10) {
    System.out.println(x);
    x++;
}
```

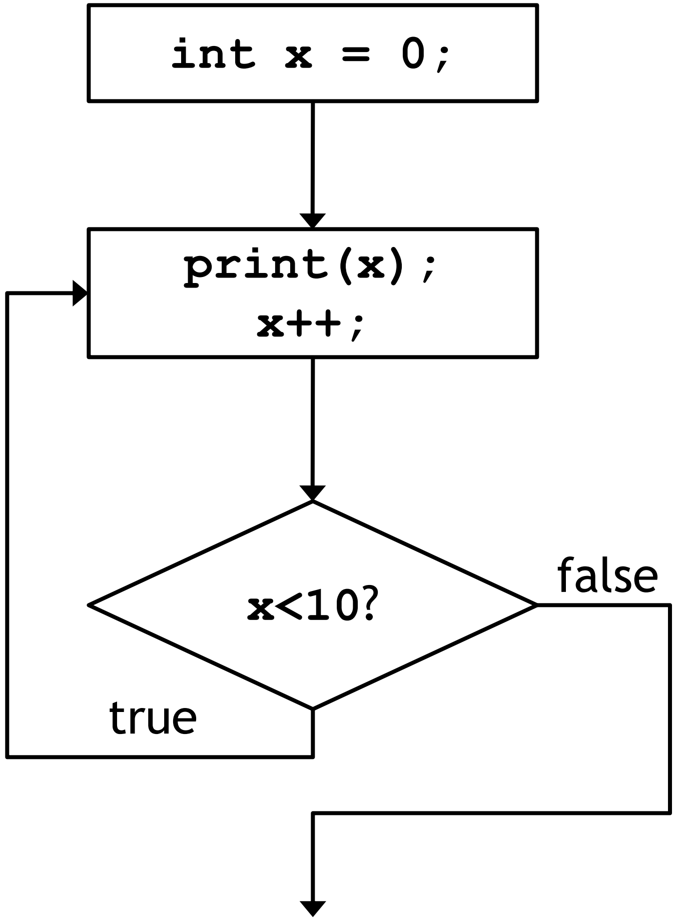
ITI - Jets © 2018 All Rights Reserved

Flow Control: Iteration – do..while loop

- The do..while loop is used when the termination condition occurs unexpectedly and is checked at the end.
- **Syntax:**

```
do
{
        …
        …
        …
}
while(boolean_condition);
```

do..while loop Example



int x = 0;

print(x);
x++;

x<10?

false

true

```
int x = 0;
do{
    System.out.println(x);
    x++;
} while (x<10);
```

ITI - Jets © 2018 All Rights Reserved

Flow Control: Iteration – for loop

- The for loop is used when the number of iterations is predetermined.
- **Syntax:**

```
for (initialization ; loop_condition ; step)
{
        …
        …
        …
}
```

```
for (int i=0 ; i<10 ; i++)
{
        …
        …
}
```

ITI - Jets © 2018 All Rights Reserved

Flow Control: Iteration –Enhanced for loop

- **The first element:**
  - is an identifier of the same type as the iterable_expression
- **The second element:**
  - is an expression specifying a collection of objects or values of the specified type.
- The enhanced loop is used when we want to iterate over arrays or collections.

```
for (type identifier : iterable_expression)
{
        // statements
}
```

Flow Control: Iteration –Enhanced for loop example

```
double[] samples = new double[50];
```

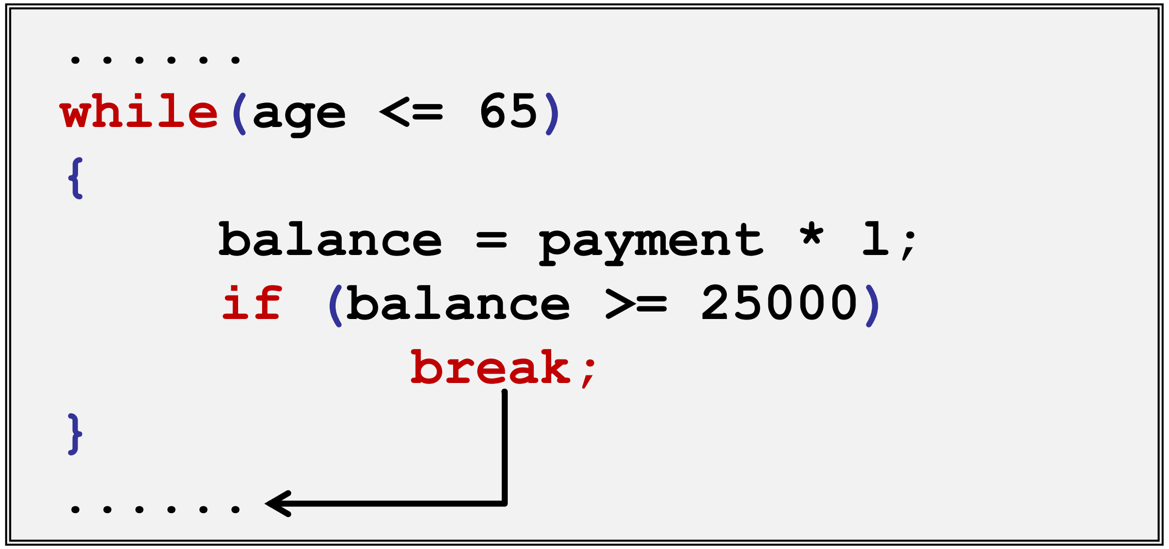
```
double average = 0.0;
for(int i=0;i<samples.length;i++)
{
     average += samples[i];
}

average /= samples.length;
```

```
double average = 0.0;
for(double value : samples)
{
     average += value;
}
average /= samples.length;
```

ITI - Jets © 2018 All Rights Reserved

The break statement

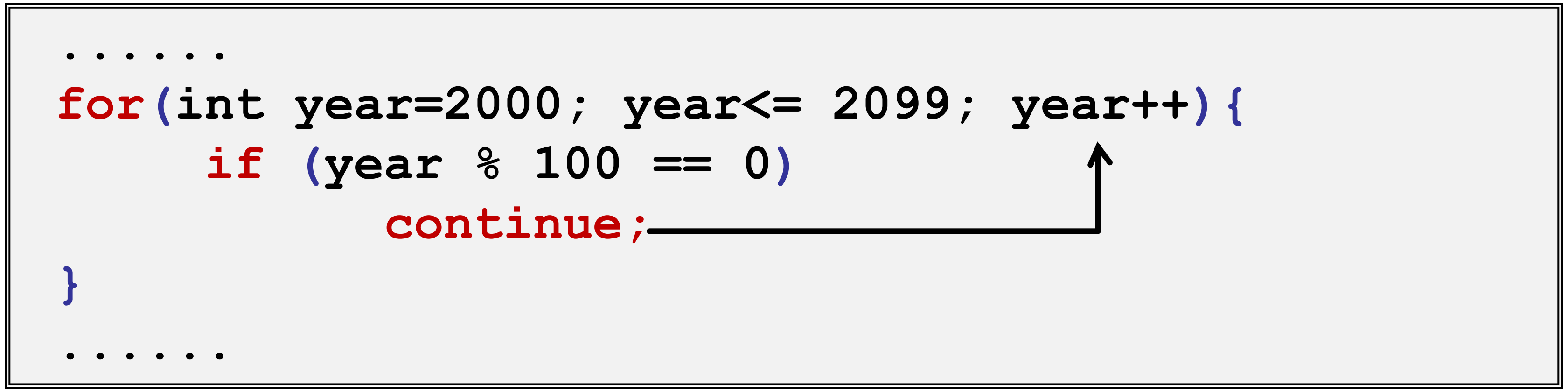
- The break statement can be used in loops or switch.
- It transfers control to the first statement after the loop body or switch body.



```
......
while(age <= 65)
{
     balance = payment * 1;
     if (balance >= 25000)
           break;
}
......
```

The `continue` statement

- The continue statement can be used Only in loops.
- Abandons the current loop iteration and jumps to the next loop iteration.



```
......
for(int year=2000; year<= 2099; year++){
     if (year % 100 == 0)
           continue;
}
......
```

Java™ Education and Technology Services

Comments in Java

- To comment a single line:

```
// write a comment here
```

- To comment multiple lines:

```
/*   comment line 1
     comment line 2
          comment line 3 */
```

Java Programming

Modifiers and Access Specifiers

Java™ Education and Technology Services

Invest In Yourself , Develop Your Career

ITI - Jets © 2018 All Rights Reserved

Modifiers and Access Specifiers

- Modifiers and Access Specifiers are a set of keywords that affect the way we work with features (classes, methods, and variables).

- The following table illustrates these keywords and how they are used.

Modifiers and Access Specifiers cont’d

Keyword	Top Level Class	Methods	Variables
public	Yes	Yes	Yes
protected	-	Yes	Yes
private	-	Yes	Yes
Final	Yes	Yes	Yes
Static	-	Yes	Yes
abstract	Yes	Yes	-

Lab Exercise

Lab Exercise 1 – Calculator

- Create a simple non-GUI Application that carries out the functionality of a basic calculator (addition, subtraction, multiplication, and division).

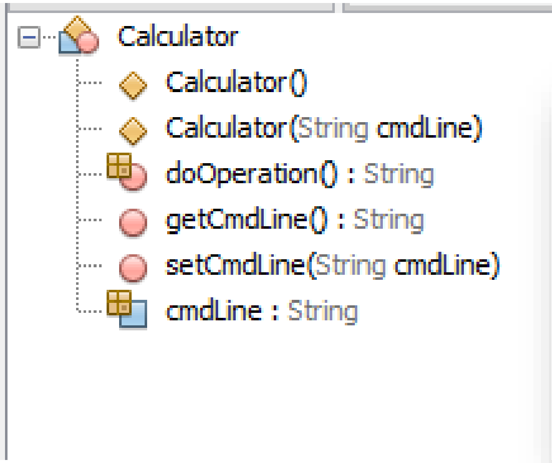
- The program, for example:

```
your Input :
70 + 30
The result is :
100
```

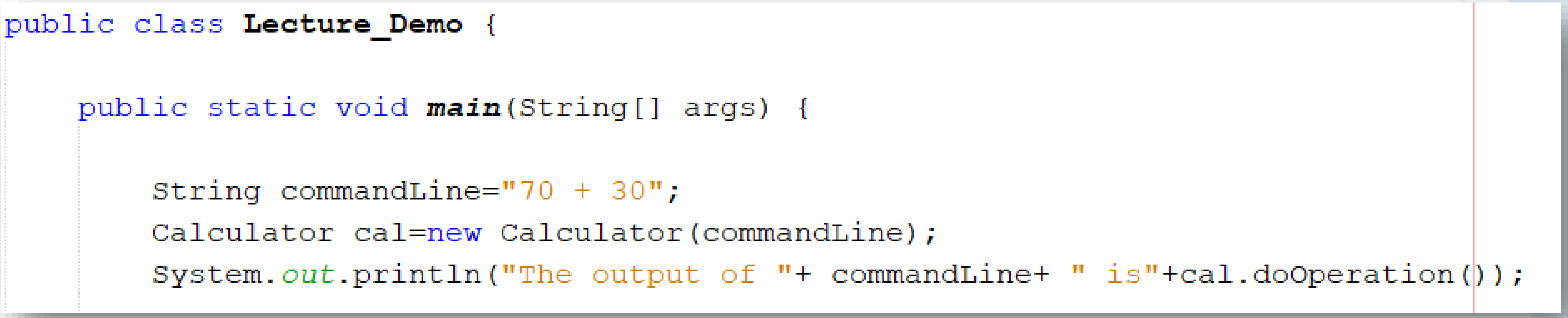
- bonus: make unlimited number of operations program:
  - 70 + 30 x 5 – 8 + 7 / 4

ITI - Jets © 2018 All Rights Reserved

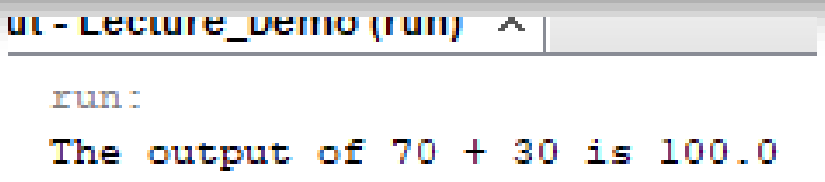
Calculator



Calculator
- Calculator()
- Calculator(String cmdLine)
- doOperation() : String
- getCmdLine() : String
- setCmdLine(String cmdLine)
- cmdLine : String

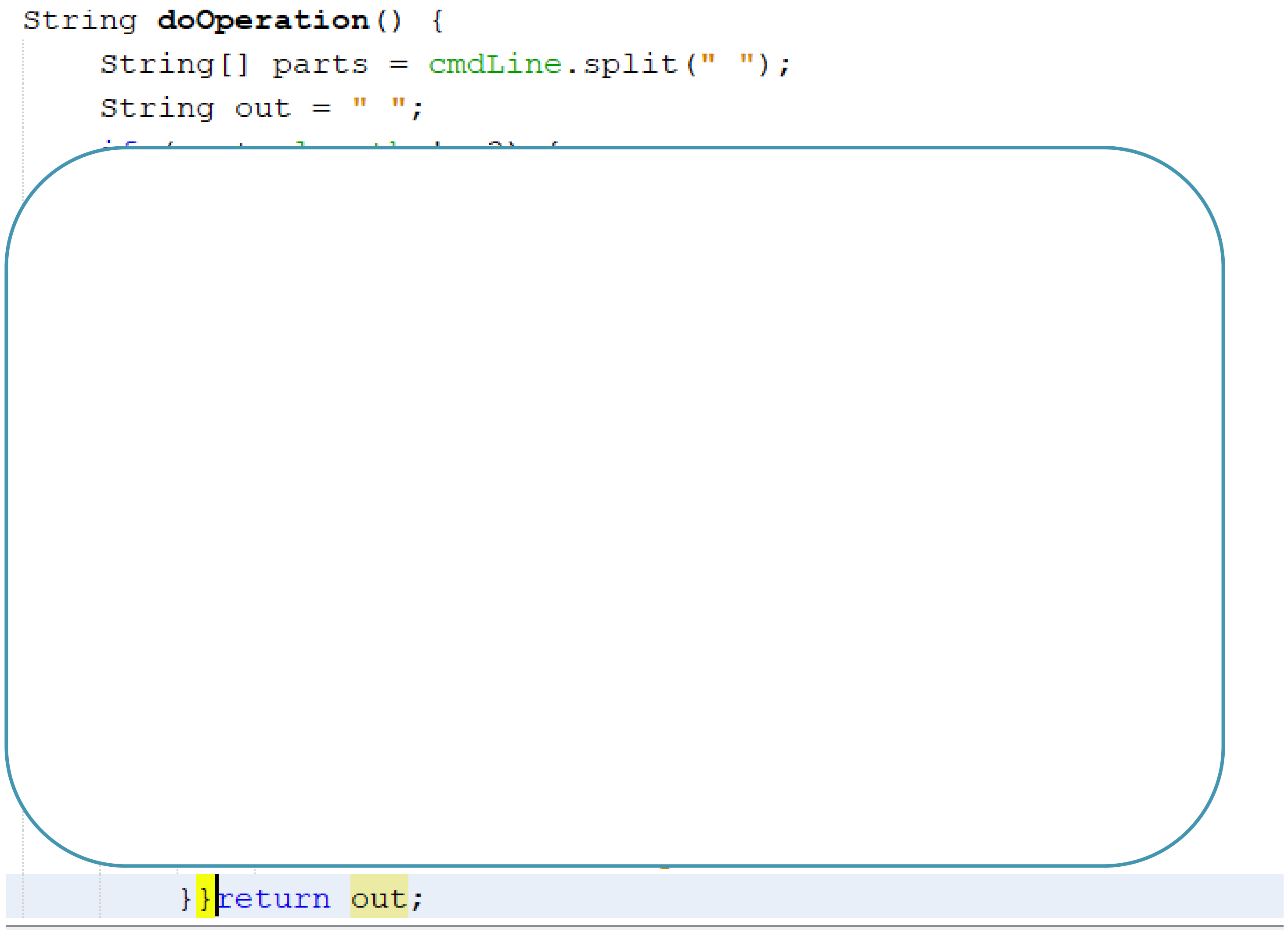


```
public class Lecture_Demo {

    public static void main(String[] args) {

        String commandLine="70 + 30";
        Calculator cal=new Calculator(commandLine);
        System.out.println("The output of "+ commandLine+ " is"+cal.doOperation());
```



```
run:
The output of 70 + 30 is 100.0
```

Calculator



```
String doOperation() {
    String[] parts = cmdLine.split(" ");
    String out = " ";
    if (parts.length != 3) {
          }}return out;
```

Lab Exercise 2 – IP Cutter

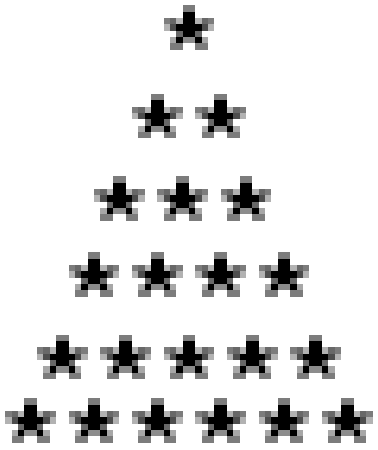
- Create a non-GUI Application that accepts a well formed IP Address in the form of a string and cuts it into separate parts based on the dot delimiter.
- Hint : split(“\\.”)
- The program, for example:

```
your Input :
163.121.12.30
The result is :
        163
        121
        12
        30
```

ITI - Jets © 2018 All Rights Reserved

Lab Exercise 3– Pattern

- Write a program that print the following pattern as user input the number of rows:



```
     ★
    ★ ★
   ★ ★ ★
  ★ ★ ★ ★
 ★ ★ ★ ★ ★
★ ★ ★ ★ ★ ★
```

ITI - Jets © 2018 All Rights Reserved